To address the computational inefficiencies of the brute-force approach, a variety of tree-based data structures have been invented. In general, these structures attempt to reduce the required number of distance calculations by efficiently encoding aggregate distance information for the sample. The basic idea is that if point is very distant from point , and a point is very close to point, then we know that points and  are very distant, without having to explicitly calculate their distance. In this way, the computational cost of a nearest neighbors’ search can be reduced to [log⁡()] or better. This is a significant improvement over brute-force for large .

An early approach to taking advantage of this aggregate information was the KD tree data structure (short for K-dimensional tree), which generalizes two-dimensional Quad-trees and 3-dimensional Oct-trees to an arbitrary number of dimensions. The KD tree is a binary tree structure which recursively partitions the parameter space along the data axes, dividing it into nested orthotropic regions into which data points are filed. The construction of a KD tree is very fast: because partitioning is performed only along the data axes, no -dimensional distances need to be computed. Once constructed, the nearest neighbor of a query point can be determined with only [log⁡()] distance computations. Though the KD tree approach is very fast for low-dimensional (<20) neighbors searches, it becomes inefficient as grows very large: this is one manifestation of the so-called “curse of dimensionality”. In scikit-learn, KD tree neighbors searches are specified using the keyword algorithm = 'kd\_tree', and are computed using the class [**KDTree**](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html#sklearn.neighbors.KDTree).